Master 2 Data Science
TD2 - Cours de Deep Learning
Réseaux convolutionnels

Repris du cours de T. Artières and S. Ayache
QARMA team - LIS lab

Janvier 2025

I. Fonction d'activation softmax et critère d'optimisation Cross entropy

On considère un réseau de neurones à K sorties pour un problème de classification multiclasses (un exemple quelconque appartient à une classe et une seule parmi les K classes données).

(1) Pour un exemple d'apprentissage (x, y) où $y \in \{0, ..., K\}$, le critère d'erreur quadratique $C_Q(w)$ est défini comme la norme de la différence entre le vecteur produit en sortie par le réseau (pour x mis en entrée), $s \in \mathbb{R}^K$, et du one hot vecteur indicateur de la classe, noté abusivement $y \in \{0, 1\}^K$, qui est un vecteur avec une seule composante à 1 (la y^{ieme}) et les autres à 0. Montrez que la dérivée du critère par rapport aux préactivations de la couche de sortie, $a \in \mathbb{R}^K$ s'écrit: $\frac{\delta C_Q(W)}{\delta a} = 2(a - y)$, en considérant que $s = g(a)$ où g est une fonction d'activation type Relu, Sigmoid, ou Tanh.

(2) Souvent pour des problèmes de classification, on utilise une fonction d'activation *softmax* sur la couche de sortie. En notant $(a_k, k = 1..K)$ les pré-activations de la dernière couche du réseau, avant la couche d'activation softmax, la fonction d'activation *softmax* calcule de nouvelles sorties égales à $s_k(x) = \frac{e^{a_k}}{\sum_{j=1...K} e^{a_j}}$. Quel est l'intérêt de cette fonction d'activation selon vous ?

(3) Montrer que la dérivée du critère par rapport aux préactivations de la dernière couche, a peut être calculée par $\frac{\delta C(W)}{\delta a} = A \times \frac{\delta C(W)}{\delta s}$ où A est une matrice que vous expliciterez ?

(4) Lorsque l'on utilise des activations softmax, on utilise souvent le critère d'entropie croisée qui est défini comme suit. Pour un exemple d'apprentissage (x, y) où $y \in \{0, ..., K\}$, le critère d'entropie est défini par: $-\sum_{k=1...K} t_k \ln s_k$ où s représente le vecteur de sorties du réseau de neurones et où t_k est égal à 1 si $y = k$ et 0 sinon. Ecrivez la dérivée de ce critère pour un exemple d'apprentissage, par rapport aux sorties du réseau de neurones. En interprétant la sortie s_k comme la probabilité a posteriori $p(y = k|x)$ montrer que minimiser le critère d'entropie croisée correspond a maximiser la vraisemblance conditionnelle des données.

II. Calcul de convolution

(1) Montrez que la taille o de la feature map en sortie d'une couche convolutionnelle 2D définie avec un filtre de taille k, un stride de taille s, un padding de taille p est égale à $o = \left\lfloor \frac{i+2p-k}{s} \right\rfloor + 1$ ou i est la taille des cartes en entrée de la couche convolutionnelle. On suppose ici que l'on s'intéresse à une dimension des cartes (largeur ou hauteur).

(2) Soient M et K les deux matrices suivantes :

$$M = \begin{bmatrix} 1 & 4 & 2 & 3 & 2 \\ 1 & 2 & 4 & 1 & 6 \\ 2 & 2 & 4 & 0 & 1 \\ 2 & 5 & 1 & 3 & 4 \\ 3 & 5 & 4 & 1 & 0 \end{bmatrix} \quad K = \begin{bmatrix} 2 & 0 & 2 \\ 3 & 5 & 1 \\ 0 & 1 & 4 \end{bmatrix}$$

Déterminez la réponse de la convolution de la matrice M par le filtre K, pour le cas où stride = 1 et half padding.

III. Etude d'une architecture CNN

Considérez l'architecture décrite ci-dessous, en PyTorch. L'argument padding='same' signifie "half padding" : lorsque le stride vaut 1, les dimensions spatiales en sortie de convolution sont égales aux feature maps d'entrée. Un biais est ajouté par défaut pour chaque neurone. Dans le cas des convolutions, il y a un biais par filtre / feature map.

```
1  import torch
2  import torch.nn as nn
3  import torch.nn.functional as F
4
5  class CustomModel(nn.Module):
6      def __init__(self, num_classes, n_input_channel,
           size_of_flattened_feature_map):
7          super(CustomModel, self).__init__()
8          self.conv1 = nn.Conv2d(n_input_channel, 32, kernel_size
               =3, stride=2, padding='same')
9          self.conv2 = nn.Conv2d(32, 64, kernel_size=5, stride=1,
                padding='same')
10         self.pool1 = nn.MaxPool2d(kernel_size=2)
11         self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride
               =1, padding='same')
12         self.pool2 = nn.MaxPool2d(kernel_size=2)
13         self.conv4 = nn.Conv2d(128, 256, kernel_size=3, stride
               =1, padding='same')
14         self.flatten = nn.Flatten()
15         self.fc1 = nn.Linear(size_of_flattened_feature_map,
               256)
16         self.dropout = nn.Dropout(0.5)
17         self.fc2 = nn.Linear(256, num_classes)
18
19
```

```
    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = self.pool1(x)
        x = F.relu(self.conv3(x))
        x = self.pool2(x)
        x = F.relu(self.conv4(x))
        x = self.flatten(x)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.softmax(self.fc2(x))
        return x
```

(1) Déterminez la taille des receptive fields (c'est à dire la taille de la fenêtre de l'image en entrée dont dépend la valeur d'un neurone = "ce que voit un neurone") des neurons des différentes couches de cette architecture ?

(2) Quelle est la taille des feature maps lorsque l'on propage une entrée dans le réseau:
   (1) Pour une donnée de taille (1, 28, 28) ?
   (2) Pour une donnée de taille (3, 64, 64) ?

(3) Combien y a-t-il de paramètres (appris, stockés) dans l'architecture suivante ?
   (1) Pour des données de taille (28, 28, 1) ?
   (2) Pour des données de taille (64, 64, 3) ?

(3) Pourquoi une telle différence de paramètres ?

(4) Que peut-on faire pour réduire le nombre de paramètres ?

IV. Calcul de gradient

On définit la sortie d'une couche convolutionnelle, de poids $W \in \mathbb{R}^{k \times k}$ appliquée à une image $X \in \mathbb{R}^{d \times d}$ par $Y = W * X$ avec un stride de 1 et où $*$ est l'opérateur de convolution.

Exprimez la quantité $\frac{\partial Y_{i,j}}{\partial W_{u,v}}$

V. Translation-équivariance

Soit X une image en entrée d'un filtre de convolution K. Pour tout $(x, y) \in \mathbb{R}^2$ on définit un opérateur de translation $T_{x,y}$ qui bouge chaque point $X_{i,j}$ dans la direction x et y. On a ainsi : $(T_{x,y}X)_{i,j} = X_{i-x,j-y}$.

Montrez que l'opérateur de convolution est translation-équivariant, i.e. c'est-à-dire qu'il est commutatif avec la translation : $(T_{x,y}X) * K = T_{x,y}(X * K)$.

VI. **Convolutions 1D (Annale Examen 2022)**

On souhaite construire un réseau de neurones pour classer des séquences collectées depuis 10 capteurs chaque heure pendant une semaine et annotées selon deux classes. On a $S = \left\{(x_i, y_i)_{i=1}^{N} \mid x_i \in \mathbb{R}^{168\times 10}, y_i \in \{-1; 1\}\right\}$.

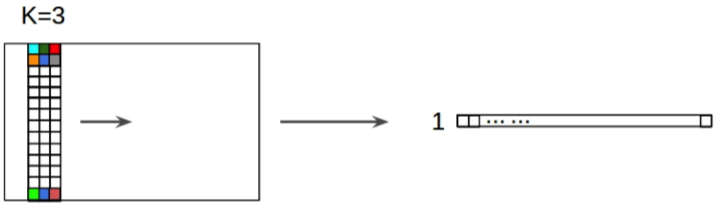

Figure 1: Couche convolutionnelle 1D. Illustration du calcul réalisé par un filtre d'une couche de convolution 1D opérant sur une séquence d'entrée de vecteurs réels de dimension 14 , où la taille du filtre est $K = 3$

On utilise dans la suite des couches de convolution 1D. Ce sont des couches de convolution dédiées à des données séquentielles. La figure 1 illustre le calcul réalisé par un filtre d'une couche convolutionnelle 1D opérant sur des séquences de longueur T de vecteurs en dimension d, i.e. $x \in \mathbb{R}^{T\times d}$. K désigne la longueur du filtre. Pour une entrée $x \in \mathbb{R}^{T\times d}$ un filtre de longueur K calcule une valeur pour chaque fenêtre de taille $K \times d$ de l'entrée (c'est à dire : le filtre est appliqué à une fenêtre temporelle de la séquence). On peut utiliser un stride qui correspond au décalage en temporel. Bien entendu on peut multiplier les filtres comme dans les couches conv2D, et "stacker" les sorties de chaque filtre pour obtenir en sortie d'une couche de convolution une séquence de vecteurs, dont la dimension temporelle dépend du stride, du padding et de la longueur de la séquence d'entrée T et dont la dimension vectorielle est le nombre de filtres utilisés.

On considère maintenant une architecture neuronale pour la tâche.

(1) Les deux premières couches du réseau sont des couches Conv1D définies pour apprendre 32 filtres de taille 5, avec biais, sans padding et stride=2, suivies d'une couche Flatten. Quel est le nombre de paramètres de ces couches ? Quelles sont les tailles des sorties de ces deux couches ?

(2) La couche suivante est une couche Dense de 64 neurones avec biais et fonction d'activation RELu. Quel est le nombre de paramètres de cette couche ?

(3) La dernière couche du réseau est une couche Dense de 2 neurones. Quelle fonction d'activation utiliser ? Quelle fonction de perte utiliser ?

(4) On constate que le réseau sur-apprend, proposez plusieurs façon d'y remédier.

(5) Pourriez vous utiliser des couches conv2D pour ce problème ? Est-ce que cela correspondrait à d'autres hypothèses sur les données ? Lesquelles ? Quelle taille de filtres de convolution serait pertinente ?